Math

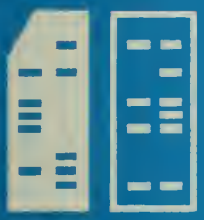
Report No. 336

AN AUTOMATIC INTEGRATOR FOR ORDINARY DIFFERENTIAL EQUATIONS FOR ILLIAC IV

by

Thomas McCarthy

June 1, 1969



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 336

AN AUTOMATIC INTEGRATOR FOR ORDINARY DIFFERENTIAL EQUATIONS FOR ILLIAC IV

by

Thomas Edmund McCarthy

June 1, 1969

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. US AF 30(602)4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, June, 1969.

ABSTRACT

This report describes the design and implementation of an automatic integrator for systems of ordinary differential equations for use on the ILLIAC IV computer system. This integrator accepts a simple statement of the differential equations and their initial values and produces an ILLIAC IV Assembly Language code which can be used to solve them. The intent is to produce a code which will be efficient on a large parallel computer.

ACKNOWLEDGMENT

The author would like to thank Professor D. J. Kuck for his advice and council on this thesis. Special words of appreciation should be given to Mr. Jacques LaFrance and Mr. Nelson Machado for their assistance in using the Translator Writing System.

The author would also like to express his appreciation for the financial support of ILLIAC IV and the Department of Computer Science for this work.

Finally, the author would also like to express his appreciation to Mrs. Kay Flessner for her speedy typing of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

Page

Figure

1. INTRODUCTION

Standard methods of integrating ordinary differential equations consist essentially of incrementing the independent variable by some small amount and then, by using past values and derivatives of the dependent variable, predicting the new value of that dependent variable. Thus the integration proceeds from the given initial value of the dependent variable to some user specified stopping point, building on itself as it goes. The same method is applied to systems of ordinary differential equations; but, of course, one cannot integrate one equation completely and then the next, since in general, the derivative of each dependent variable will depend on the other dependent variables of the system. Each equation must be integrated at a given value of the independent variable before going to the next step.

For this reason, to perform an integration of a system on a large parallel computer like ILLIAC IV, it is desirable to perform the integration on all of the equations simultaneously. Difficulties arise when the equations are not identical in form; and, of course, in general they are not. Integration methods consist of one or more function evaluations; that is, substituting the present value of the dependent variable into the equation to find its derivative at the given point. This is the part that is most difficult to do simultaneously with dissimilar equations, and it is the part which usually accounts for much of the computation time, even on a serial machine.

The following example will illustrate some of the problems to be solved:

$$y_1' = a\, y_1\, y_2 + b\, y_1 + c\, y_1\, y_1 / y_2$$
$$y_2' = d\, y_2 + e\, y_1\, y_2 + f\, y_1\, y_2$$

The logical way to proceed would seem to be to try to minimize the number of nonparallel arithmetic operations which must be done. For an example of how this might be done, consider a two processing element machine with each processor working on a single equation. Assume that by allocating storage properly and by using some indexing (to be explained later), it is possible to access constants and different dependent variables simultaneously. The operation sequence would then be as follows (parentheses indicate memory access, I indicates processor idle, arithmetic operations indicated by their symbols).

P1 (a) x x (b) x I + (c) x x ÷ +
P2 (d) x I (e) x x + (f) x x I +

If the terms can be rearranged so that the first term of the first equation and the second term of the second equation can be done at the same time, some more savings can be gained.

P1 (a) x x (b) x + (c) x x ÷ +
P2 (e) x x (d) x + (f) x x I +

Thus more operations are done in parallel and the processors are idle fewer times. With larger systems of equations there are more terms that can be done in parallel and more comparisons to be made. With rearrangement, unmatched terms could also be done partly in parallel. With the larger number of terms in big systems, it is reasonable to expect that the relative savings will be greater than those in this example.

Another problem would arise if one of the equations had significantly more terms than the other. Some method of sharing the work between processors should be used. As the system of equations gets larger in number of terms and number of equations, all of these problems become more and more complex.

The translator-integrator program described in this thesis tries to do these types of optimizations as well as compiling the Assembly code to evaluate the input equations. The translator-integrator (which will be referred to as the integrator in the following pages - the section designated should be clear from the context) has associated with it some skeleton forms of ILLIAC IV Assembly Language programs to solve ordinary differential equations by different numerical methods. These, along with the compiled code and generated storage pseudo-ops allows the program to produce a completed Assembly code to solve the equations.

The translator-integrator system itself is an Algol program which would be executed on the Burroughs 6500 in the ILLIAC IV system. A schematic of the integrator is shown in Figure 1.

Section 2 describes the integrator from the user's point of view. Section 3 is a discussion of the implementation techniques and numerical methods used. Section 4 discusses the capabilities of this system and the further work which might be done.

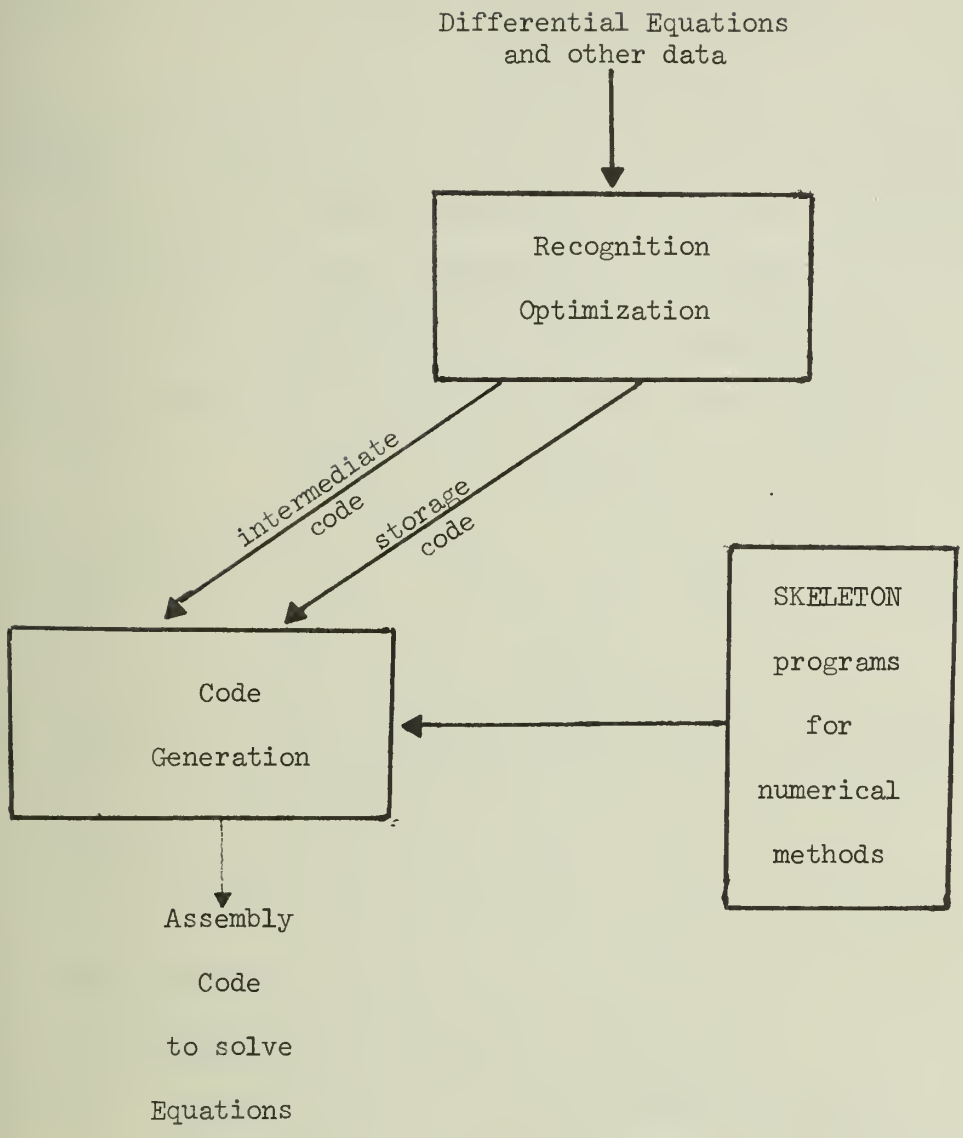


Figure 1. Schematic Diagram of Integrator System

2. USE OF THE INTEGRATOR SYSTEM

The primary object of any automatic integration system is to provide the user with a device to solve his equations with a minimum of effort and analysis. From the user's point of view, the integrator input is as defined in Backus Normal Form (BNF) in Figure 2. Most of this should be self-explanatory. Dependent variables are expressed as Y(N); their derivatives as Z(N), and the independent variable as T. Each differential equation is written as

$$Z(N) = f_n\ (Y(1),\ Y(2),\ \ldots,\ T)$$

The functions can be almost any Algol expression (an obvious exception is an if clause) as defined by the BNF and is written in B6500 Algol notation. It should be noted that constants whose values can be assigned later are allowed in these expressions. The initial values of the dependent variables and the starting and final values of the independent variable are then given. These are real numbers expressed in Algol notation. All numbers except indexes are treated as 64-bit floating point numbers by the integrator. It is not necessary to terminate whole numbers with a decimal point to insure that they are floating point. TO and TF are the starting and ending values of T, the independent variable.

The user can also either specify a stepsize to be used or specify a desired maximum error which the integrator will try to stay

```
<program> ::= <differential equations> <initial values> <constant declarations>
                    <other statements> END
<differential equations> ::= Z( <integer> ) = <express i on> ; |
                    <differential equations>  Z( <integer> )= <expression> ;
<initial values> ::= ZO( <integer> ) = <real number> ; |
                    <initial values> ZO( <integer> ) = <real number> ;
<constant declarations> ::= <identifier> = <real number> ; |
                    <constant declarations>  <identifier> = <real number> ;
<other statements> ::= <statement> ; | <other statements>  <statement> ;
<statement> ::= TO = <real number> |  TF = <real number> | STEP = <real number> |
                    ERROR = <real number>
<expression> ::= <term> | <add operator> <term> | <expression> <add operator>
                    <term>
<add operator> ::= + | -
<term> ::= <factor>  | <term> <multiply operator> <factor>
<multiply operator> ::= x | /
<factor> ::= <primary> | <factor> * <primary>
<primary> ::= <real number> | <identifier> | Y( <integer> )  |  T  |
                    <funid> ( <expression> )  |  (<expression>)
<funid> ::=   SIN  |  COS  |  EXP  |  LOG
```

Figure 2. Input Language Description in BNF

within. Also any constants which were used in the equations should be defined. The program input is terminated with an END.

In general the input is free form with semicolons separating statements. In fact, a missing semicolon will not always cause an error, but it is suggested that the user try to make the input no more difficult to recognize than necessary. Identifiers are limited to six characters; longer ones are truncated on the right. Numbers are limited to eighteen characters including sign and exponent. All blanks are ignored.

From this minimal amount of information an Assembly Language Code will be produced and placed as a file on disk. If some information, such as an initial value or constant declaration is omitted, a code will still be produced with zeros in the appropriate locations. In Figure 3, a sample program for the example discussed in the introduction is shown as it would be inputted to the translator. A larger example, complete with output, is given in the Appendix.

The integrator also has some control words which can be used if desired. These are entered at the beginning of the input. The string of control words separated from each other by at least one space must be preceded by a '#' and a space. The control words are as follows:

TEST - Do not generate Assembly Language Code. This control word is used when the user is only interested in knowing what efficiency can be achieved with his equations.

```
          # RUNGE  ERRORCHECK     ;
Z(1)=  AxY(1)xY(2) + BxY(1)  +  CxY(1)xY(1)/Y(2)  ;
Z(2)=  DxY(2)+ExY(1)xY(2) + FxY(1)xY(2)  ;
ZO(1)=  0  ;
ZO(2)=  2.3  ;
A = 3.456  ;
B = 0  ;
C = 1  ;
D = 3.45677483645@-09  ;
E = 4.9 ;  F =  .33  ;
TO = 0.0  ;  TF = 1.0  ;
ERROR = .0001  ;
END
```

Figure 3. Sample Input

RUNGE - Use a Fourth-Order Runge Kutta method to integrate the equations. This is the default option.

CORPRE - Use a Corrector-Predictor method to integrate the equations. This is a more complicated method but could be used on more difficult equations.

ERRORCHECK - Estimate truncation error and adjust stepsize accordingly. This control word should be used when an error bound is to be put on the results as previously described.

CONSTMAP - This prints out a chart showing the location of constants in PE memory so that they can be changed without reexecuting the integrator.

FUNMODE - If the terms in the differential equations are predominantly made up of parenthesized expressions or function calls (i.e., SIN, COS, etc.) or both, then this option should be turned on to improve the efficiency of the code generated.

PATTERN - The integrator makes some of its decisions based on the supposition that the terms in the first few equations are representative types for the whole system. If this is not so for a particular system, the efficiency may be improved by using this control word. When this option is used, the first equation should contain terms typical of the system and in about the same ratio to the other terms in the equation as they occur in the system. No code will be generated for this equation, but it will aid the integrator in allocating the work to be done.

The Assembly Language Program which is produced also has some comments in it so that with the information provided by the CONSTMAP option of the integrator, the user should be able to make small changes, such as the values of constants or initial values, without rerunning the program if desired. An alternative to this would be to

read in data at execution time. When the input conventions for ILLIAC IV are defined, the integrator can easily be modified to load the data into the correct place.

3. IMPLEMENTATION

3.1 An Overview of the System

3.1.1 The Recognizer

The recognizing part of the integrator program was written using the ILLIAC IV Translator Writing System (TWS). The input to one of the TWS programs is a Backus Normal Form (BNF) description of the integrator input language syntax. This program, which runs on the Burroughs 5500, generates an Algol scanner and parser which can be used to recognize programs written in the language defined by the input BNF. An action number can be associated with each production to execute a designated semantic routine. A TWS program is also provided to concatenate these semantic routines to the parser. While this program also provides several built-in routines designed to help write compilers, few of these were actually used due to the specialized nature of the integrator language. All of the analysis, reduction and code generation for the integrator is done within the semantic routines.

The actual input to the TWS programs is somewhat different than that given in Figure 2. Action numbers are required, and the BNF description is slightly different to aid the syntactic analysis.

3.1.2 The Semantic Routines

Before the parsing of the input begins, a short, fast prepass is made in the initial semantic routine to count the number of equations

and number of terms (by counting arithmetic operators). Since the input to the integrator may be quite large, it is necessary to do optimization and the related compilation and storage allocation as the program proceeds, so as to limit the amount of B6500 storage needed. The information gleaned from this prepass is needed to do this dynamic analysis.

As the input differential equations are parsed, an intermediate code for evaluating them is generated. The optimization is then performed on a term by term basis on this intermediate code. Storage assignment begins at the first processing element and then proceeds according to the types of terms and equations involved in the problem. Other types of statements such as initial values and constant declarations are recognized, and the information is saved. When the entire input has been recognized, all the necessary information for Assembly code generation has been formed. At this point the appropriate skeleton ILLIAC IV integration program is selected, and the storage declarations and the code for evaluating the equations are inserted in the proper places to form a completed Assembly Language Code. This is left as a disk file which can be assembled to machine code.

3.2 Methods for Improving Parallelism

3.2.1 Possible Instructions and Operands

The instructions used to evaluate an equation are a small and standard set; for example, load register, add to register, multiply,

store, etc. Thus the operands that will be used also constitute a fairly limited set. In general, they can be divided into the following classifications:

1. CONSTANTS - These values do not change during the integration. They may have been declared either as literals or by constant declaration statements.

2. DEPENDENT VARIABLE - These have an index value associated with them declared in input language as Y (<index>).

3. FUNCTION CALLS - These include any of the explicitly allowed functions such as SIN and COS.

4. INDEPENDENT VARIABLE - This is designated by T in the input language.

5. REGISTER NAME - These occur in register to register instructions used for temporary storage of data.

6. NO OPERAND - This is for such instructions as change the sign of the A register (CHSA).

3.2.2 Doing Similar Instructions in Parallel

First consider the same instruction operating on different operands which fall into the same operand class. Methods must be developed for doing these in parallel.

When the operand is a constant, as many consecutive rows of PE memory as needed are blocked off. When the Assembly code is generated, instructions which access constants are given operand fields such that they point to consecutive rows in this array. Each processor can have a different value in its memory position in the constant row.

It is a simple matter for the integrator to take the values defined as literals or by constant declarations and insert them in the correct row and column so they are accessed by the correct instruction.

The values of the dependent variables are not stored in the same way as constants since they must be updated constantly throughout the execution of the integration to be performed. The values of the independent variables that are needed are kept in each PE in a column. The way to access different Y's in parallel is to associate a tag row with each access in a way similar to that used for constants. In this case the tag row is loaded into the X register when needed and a load modified by index is made from the independent variable storage area. While this costs an instruction to load the X register, it really only costs $^{1}/_{m}$, where m is the number of equations, since the indexes for all of the variables are loaded at once. Also in a serial machine these variables would often be declared as an indexed array.

At the present stage of design, it is not possible to execute different function calls in parallel, since a function call is in reality a reasonably large set of instructions to be inserted in this position, and the processing elements must execute the same instructions. Since most functions form some sort of polynomial approximation, it would be a reasonable extension of the system to have the integrator provide its own library functions which have been written so they are as parallel as possible and thus gain some speed.

Since there is only one independent variable, instructions with it as an operand can always be performed in parallel. The value is

kept in all PE's in the same memory row and updated when appropriate. No operand instructions can clearly always be done in parallel.

Instructions with different register names as operands clearly cannot be done in parallel. Since these only occur in temporary storage manipulation according to a well-defined algorithm, similar terms will probably access the same registers anyway. Also, due to the method used for sharing terms described in a later section, little temporary storage will be used.

3.2.3 Term by Term Analysis

In order to compare equations to see what can be done in parallel, it is necessary to break them down in to more convenient and hopefully more similar parts. The unit selected was that of a term since these can be shared to other processors and then recombined with just an addition operation. The expressions representing the differential equations are only broken down into the first level of terms; that is, terms inside parentheses are not separated. Some examples are listed below. The expressions on the left are broken down into the terms (separated by commas) on the right. The expression can be reformed by just adding the terms on the right.

A + B + C ⟶ A, B, C

A + B - C ⟶ A, B, -C

A + (B-C) ⟶ A, (B-C)

AxY(1) + BxY(2) x Y(3) + SIN (Y(4)) ⟶

AxY(1), BxY(2) x Y(3), SIN (Y(4))

Of course, one can certainly write separate terms according to this rule which differ tremendously, but it is reasonable to expect that in a large system of equations many terms will be similar. Further discussion of the integrator should make obvious the convenience of this division. Thus if all of the instructions in two terms are the same and the operands fall into the same classes as defined by previous remarks, then these terms can be done in separate PE's simultaneously.

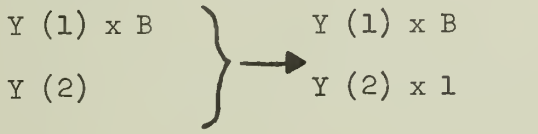
3.2.4 Inserting Dummy Instructions

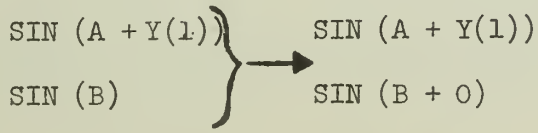
In actuality it cannot be assumed that all terms will be perfect matches with other terms. Consider the following terms:

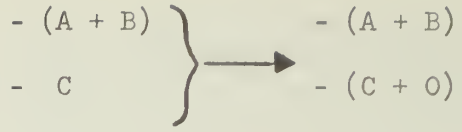
Ax Y (1) x Y (3)

Bx Y (3)

They differ only in one multiply; and if no other better pairing is available, this can be treated as Bx Y (3) x 1. In actuality it is treated as Bx Y (3) x Y (0) where Y (0) always contains a one. Thus these terms can be paired. Other dummy instructions can be inserted as follows:







Inserting dummy operands for instructions involving other than constants and dependent variables does not appear worthwhile since extra instructions would be needed. Since a function call usually involves several instructions, it is not advisable to have a dummy call unless no better match can be found.

3.2.5 Sharing Terms

As previously mentioned, one of the important problems is that the equations may vary greatly in the number of terms in each. Some method is needed to share the work more or less equally between processors. This is the place where the information gained in the prepass becomes necessary. A limit is set on the number of terms to be evaluated in each PE according to the following equation:

$$\text{maximum number of terms / PE} = \left\lceil \frac{\text{total number of terms}}{\text{total number of PE's}} \right\rceil$$

The square brackets indicate the smallest integer greater than the enclosed value. This limit serves to distribute the terms equally across the PE's. When the optimization has assigned the maximum number of terms to a PE, it then skips to the next. This requires that a tag be associated with each term indicating which equation it belongs to.

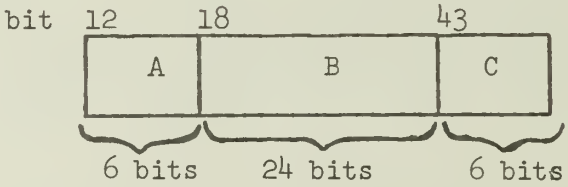
After a term has been evaluated, it is added to a scratch memory location in the same PE. This location depends on the tag so that at the end of the evaluation, there may be several separate partial sums in a particular PE. After all partial sums are formed, routes and adds are performed which combine partial sums and return them to the correct PE's. Also, if the number of PE's is much greater than (i.e., more than twice) the number of equations, the integration procedures other than function evaluation are assigned to every other one or two PE's sequentially so that the amount of routing necessary will be reduced. Assigning the PE's sequentially assures that the routes will be small since the terms are also assigned to the PE's more or less sequentially within the range of the optimization routine.

3.3 The Optimization Routine

The optimization routine is the procedure which is called when a term has been recognized. Its purpose is to determine what term this just recognized term can be done in parallel with. Flowcharts are given in Figures 4-6. There are three basic stacks. COMPSTK is the stack containing the intermediate code for the term just recognized. CODESTK contains the intermediate code for the terms which are to be shared over the processors as previously discussed. EXCODESTK contains the code for the terms which cannot be matched with those in CODESTK. In the normal state of operation, terms which contain function calls and parenthesized expressions are put in this stack since they are expected

to be the exception and not the rule. This can be changed by the user if desired with the control word FUNMODE. Optimization does occur within EXCODESTK as well as CODESTK.

Basically the optimization routine executes a series of calls to a comparison routine with different arguments. The arguments tell this comparison routine which stacks to compare and how many dummy instructions it can insert in each in order to make terms match. The flowcharts show the sequence of these calls. The purpose of the particular order is to find the match which requires the least overhead first. When the maximum number of terms allowed in CODESTK is reached, the data for storage generation is extracted for that PE, and the program proceeds to the next. CODESTK and EXCODESTK are overwritten for each PE since the basic instruction and operand will be the same if a comparison is achieved. Only the storage information contained in the intermediate code is different. The pattern for the intermediate code given below illustrates this.



A - code number indicating the instruction (LDA, ADRN, . . .)

B - storage information

C - type of operand

If C is	→	then B contains
No operand	→	0
Constant	→	symbol table pointer
Dependent Variable	→	index value
Independent Variable	→	0
Function Call	→	Name of function
Register to Register	→	Source register name
Arithmetic computation temporary storage	→	index within storage
Exponentiation temporary storage	→	index within storage

The code remaining when the final equation has been optimized is used to form the ILLIAC IV code itself.

An example of the previously discussed case follows:

EQUATIONS:

Z(1) = AxY(1) x Y(2) + BxY(1) + CxY(1) x Y(2) / Y(2)

Z(2) = DxY(2) + ExY(2) x Y(2) + FxY(1) x Y(2)

DETERMINED IN PASS 1:

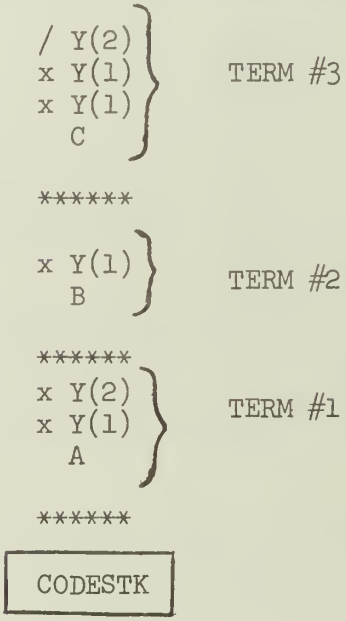
MAXIMUM TERMS PER PE = 3

NUMBER OF EQUATIONS = 2

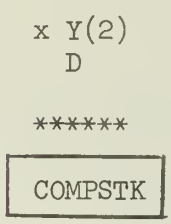
NOTE: ASSUMING A 2 PE SYSTEM

PASS 2:

1. FIRST THREE TERMS ARE READ IN TO FILL UP CODESTK. STORAGE ALLOCATION FOR THE FIRST PE IS DONE.



2. READ NEXT TERM IN INPUT AND PUT IN COMPSTK.

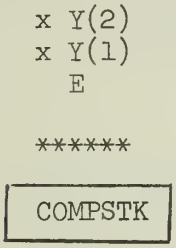


3. COMPARE COMPSTK TO CODESTK WITH NO DUMMY INSERTIONS ALLOWED.

4. FIND A MATCH WITH TERM 2.

5. COMPSTK IS OVERWRITTEN INTO CODESTK. FLAGS ARE SET INDICATING THAT THIS TERM IS TAKEN.

6. NEXT TERM IS READ INTO COMPSTK.

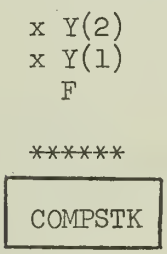


7. COMPARE COMPSTK TO CODESTK WITH NO DUMMY INSERTIONS ALLOWED.

8. FIND A MATCH WITH TERM #1.

9. OVERWRITE AS IN STEP 5.

10. READ IN NEXT TERM.



11. COMPARE WITH REMAINING TERM IN CODESTK - NO INSERTIONS.

12. FAILURE - TRY TO COMPARE ALLOWING ONE INSERTION

13. SUCCESS

```
/ Y(0)
x Y(2)
x Y(1)
  F

******
```

Note: Y(0) always contains a 1.

COMPSTK

14. OVERWRITE AS IN STEP 5.

```
/ Y(0)
x Y(2)
x Y(1)
  F

******

x Y(2)
  D

******

x Y(2)
x Y(1)
  E

******
```

CODESTK

15. USE CODESTK TO ALLOCATE STORAGE FOR THIS PE.

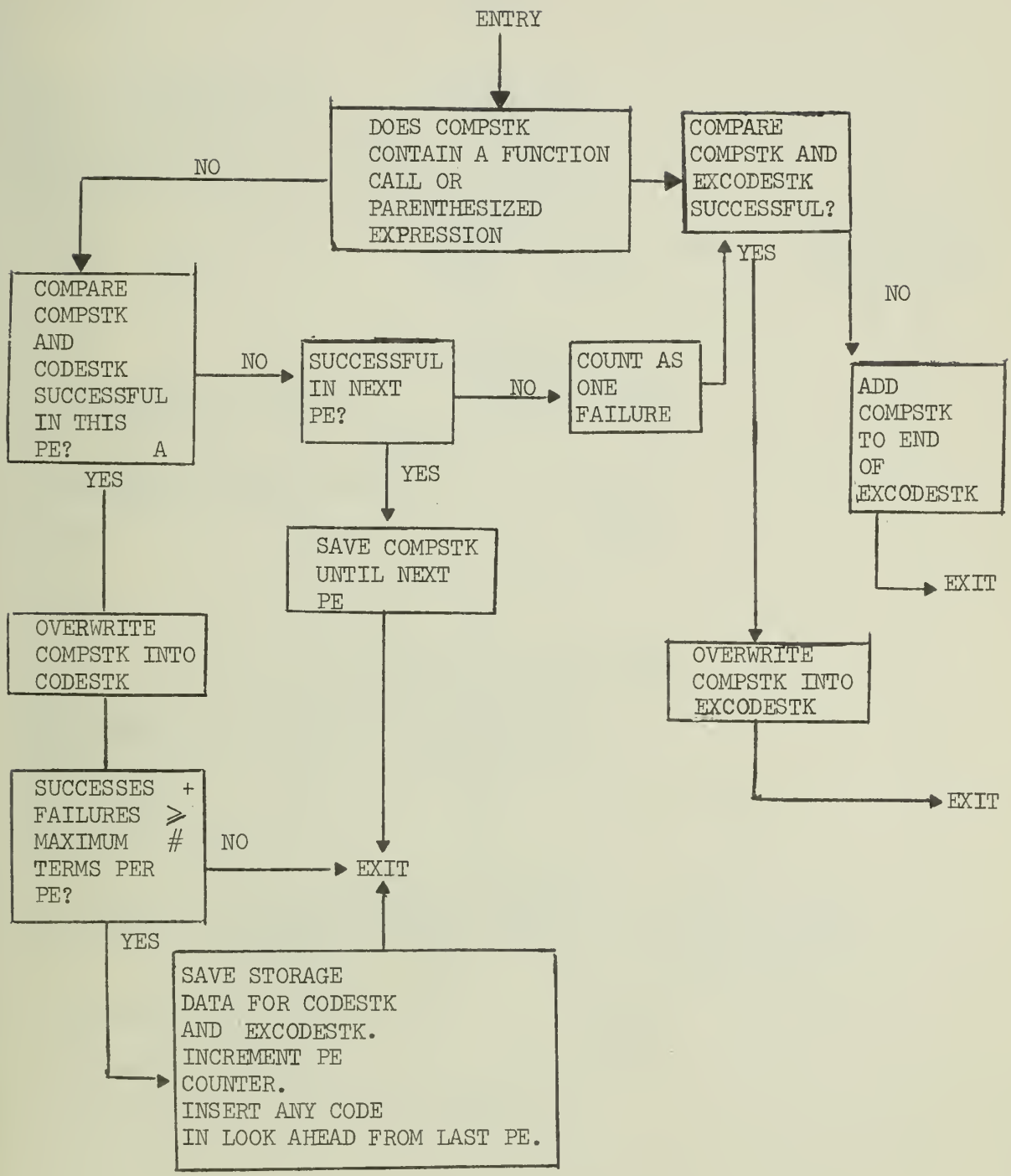


Figure 4. Optimization Routine

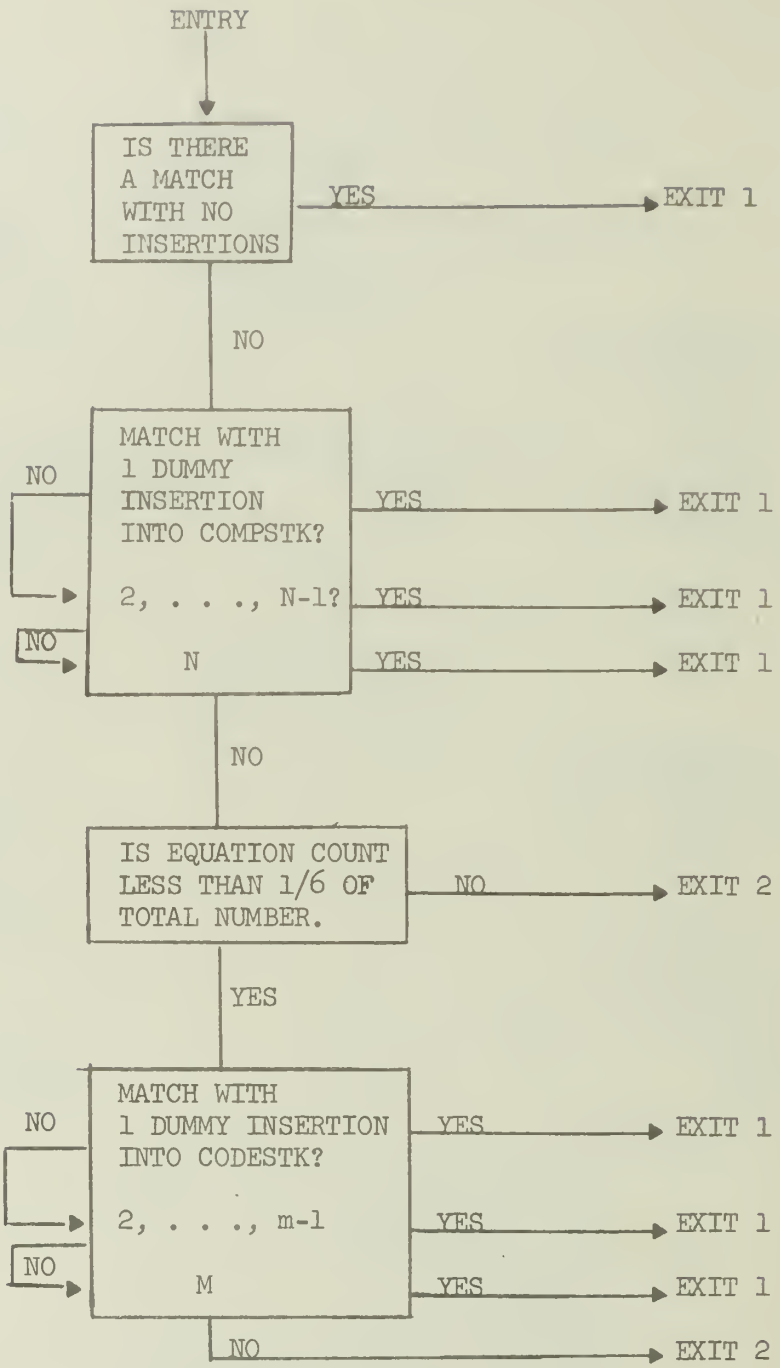


Figure 5. Comparing Compstk and Codestk BLOCK A in PREVIOUS FLOWCHART

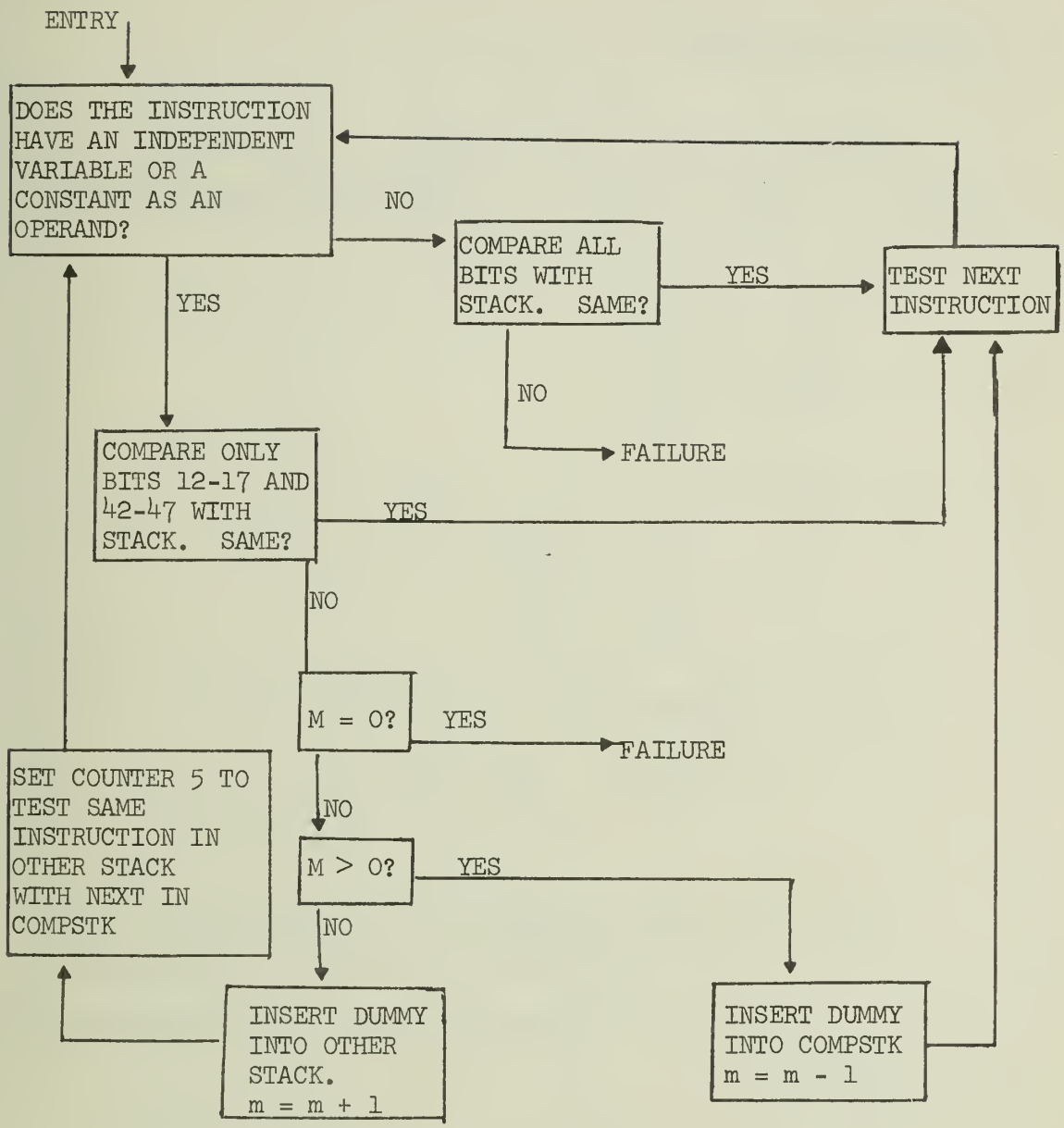


Figure 6. Routine to Compare COMPSTK with Another Stack. M is the number of insertions of dummy instructions allowed.

3.4 Methods of Integration

At present there are two integration methods available for the integrator. These are Fourth Order Runge Kutta and a corrector-predictor method investigated by Nordsieck.

The standard Fourth Order Runge Kutta method is used. This is as in the equations below.

$$K_1 = F\ (Y_n,\ T)$$

$$K_2 = F\ (Y_n + \tfrac{1}{2} HK_1,\ T + H/2)$$

$$K_3 = F\ (Y_n + \tfrac{1}{2} HK_2,\ T + H/2)$$

$$K_4 = F\ (Y_n + HK_3,\ T + H)$$

$$Y_{n+1} = Y_n + \tfrac{1}{6} H\ (K_1 + 2K_2 + 2K_3 + K_4)$$

If automatic stepsize adjustment is called for, spare PE's, which are not used for the integration, are used to do the integration at twice the stepsize. The truncation error can then be estimated from the following formula.

$$E_t = Kh^5 = \frac{32}{31}\left[y_m(h/2 - y_m(h)\right]$$

Stepsizes are always changed by a factor of two and certain limitations are put on the changing frequency and minimum size of the step.

The corrector-predictor method is a multistep method and thus requires another method to initialize it. The midpoint rule is used in the integrator. When the starting values y_{n-1} and y_{n-2} are found, they are premultiplied by a matrix as follows:

$$\underline{a}_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3/4 & -1 & 1/4 \\ 0 & 1/6 & -1/3 & 1/6 \end{bmatrix} \begin{bmatrix} y_n \\ hf(y_n) \\ hf(y_{n-1}) \\ hf(y_{n-2}) \end{bmatrix}$$

It can be shown that the resulting $\underline{a}_n$ is equivalent to

$$\left[y_n,\ hy_n',\ h^2y_n''/2,\ h^3y_n'''/6 \right]^T$$

The formulas for the integration method are then:

$$\underline{a}_{n+1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \underline{a}_n + \begin{bmatrix} -3/8 \\ -1 \\ -3/4 \\ -1/6 \end{bmatrix}$$

where $F = h\,y_n' - h\,f(y_n)$. There are several reasons for using this method. For reasons which are beyond the scope of this explanation,

the method minimizes roundoff error. Also, due to the Pascal triangle form of the matrix, the matrix multiplication can be reduced to a series of a few additions thus speeding up the method. The biggest advantage is that changing the stepsize does not require restarting procedures. The vector $\underline{a}_n$ simply is multiplied by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & \alpha^2 & 0 \\ 0 & 0 & 0 & \alpha^3 \end{bmatrix} \qquad \text{where } \alpha = \frac{h_{old}}{h_{new}}$$

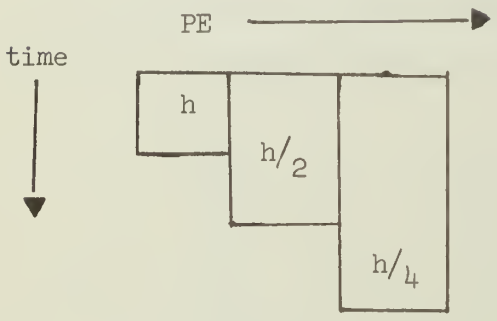
Each of these methods is stored as a skeleton ILLIAC IV program which is used to form the complete integrator output.

4. EVALUATION AND SUGGESTIONS FOR FURTHER DEVELOPMENT OF THE INTEGRATOR

As might well be expected, the efficiency of the ILLIAC IV program generated by the integrator system varies greatly on the type of input. One would expect it would not do as well on equations with highly complicated and diverse terms, and this proved to be true. Equations such as those involved in chemical kinetics problems where the terms differ only in two or three multiplys were handled very well, approaching the maximum efficiency for this algorithm.

The integrator is also not well suited to small systems of equations. There are many reasons for this. Conventional integration methods are rigidly sequential, so the work other than function evaluating can only use as many PE's as there are equations. Attempts to separate methods into usable parallel forms have been unsuccessful to date. This method thus leaves many of the PE's unoccupied when doing the method calculations for small systems. Also, with smaller systems, the ratio of these calculations to the function evaluation is high and so the overall integration becomes inefficient. One way of improving this might be to use some extrapolation method, such as Richardson's. A lower order, and hence faster method, could be used to do the integration, thus decreasing the integration-to-function evaluation ratio. Since this involves doing the integration at different stepsizes, more of the PE's could be performing useful work in parallel. However, due to the sequential nature of the methods, there would still be a lot

of PE's empty. For example, consider a three step extrapolation method halving the stepsize for each integration. The PE versus time graph would look as follows:



This represents the time involved in all the integration and function evaluation except for the evaluation of the extrapolation formula. In this case 40 per cent of the machine time available is being used. In the best case with only two stepsizes, 25 per cent of the machine time is unused, and the savings gained by an extrapolation from two cases would probably be negligible anyway. One is tempted to move part of the process time from the smaller stepsize to the empty spaces in the larger, but this cannot be done due to the aforementioned sequential nature of the methods. The starting values to begin the integration in the middle are not available until the integration has been performed up to that point. For these reasons extrapolation methods seem of doubtful value. There are other problems which affect small sets of equations. Since there are few terms, it can be expected that, on the average, there will only be a few that can be done in parallel. Thus the function evaluation may be inefficient as well.

With larger systems the results are significantly better. There are fewer wasted PE's during the integration operations. There are more terms to be compared and to fill up the machine. It cannot, however, be guaranteed that 100 per cent efficiency will be obtained even if the maximum number of terms can be done in parallel. This is due to the fact that if the number of terms is not an even multiple of 64, another term evaluation must be performed even if there is only one extra term. The following chart illustrates the maximum efficiency for different numbers of terms.

Maximum terms per PE	Worst case #terms	efficiency	Average #terms	efficiency
1	1	15%	32	50%
2	65	51%	96	75%
4	193	76%	224	87%
8	449	87%	480	94%
20	1217	95%	1248	98%

This clearly indicates some of the improvement with larger systems. Breaking up large terms into smaller ones might improve this although there will be some added overhead in keeping track of the extra terms and recombining to obtain the results.

Another item which causes inefficiency is the routing which must be done to recombine the partial sums after their evaluation. If the equations are roughly all the same length, the routing will be minimal. Strange distributions of long and short equations might greatly

increase the routing. An example might be a set where the first thirty equations have two terms each and the last thirty have twenty terms each. If these same equations were alternated - one short, one long, one short - the routing would be reasonable. It would be desirable to have the integrator system do this type of exchange before analyzing the equations. Of course, an intelligent user could do the same thing.

Another factor is the depth of analysis used to find terms that can be done in parallel. Highly complex terms are difficult to handle. Consider the following:

```
Bx (Ax SIN (Y(1)) / 2 + Y (3) * 3)
(Dx Y (2) - SIN (Y (6)) x Cx Y (2)) * 6
```

It is clear that many of these terms could be done in parallel, but highly sophisticated techniques would be needed to do this automatically. To do this high a level analysis on every term might take enormous amounts of computer time and might only be worthwhile on smaller systems.

As previously mentioned, another useful addition would be to have routines written so that a sine and a tangent, for example, could be done simultaneously. This would seem to be possible since the common methods for obtaining these functions are by using some approximating polynomial. It might be possible to use the same form polynomial with different constants to get different functions. It should be noted that sine and cosine are trivial to do in parallel since $\cos x = \sin (x + \pi/2)$.

It goes without saying that an integration method which is less sequential would be valuable in improving this system.

In summary, the integrator system works reasonably well on large systems of equations although there is still some improvement possible. For small systems it appears that a different algorithm would need to be used to get really good results.

APPENDIX A

SAMPLE INPUT PROBLEM FOR THE INTEGRATOR-TRANSLATOR SYSTEM (OUTPUT LISTING)

ILLIAC IV TRANSLATOR WRITI G SYSTEM

GODES COMPILER - VERSION 1 0 PROGRAM G /ODES .

APRIL 30, 1969 AT 2013 : 8.9

```
$LIST RSWD                                                                    *    200    1
     #  PATTERN ;                                                             *    300    2
Z(0) = A×Y(0)×Y(0)+B×Y(0)×Y(0)+C×Y(0)×Y(0) ;                                  *    400    3
Z(1)=K1×Y(8)+A1×Y(1)+A6×Y(3)×Y(30)+A2×Y(1)+A3×Y(2)+A4×Y(7)+A5×Y(5)+6003;     *    500    4
Z(2)=B3×Y(1)×Y(9)+B2×Y(2)+B3×Y(2)+B4×Y(2)×Y(9)+B5×Y(2)×Y(9)+K2×Y(5);        *    600    5
Z(3)=C1×Y(3)×Y(9)+C2×Y(3)×Y(9)+C3×Y(3)+K3×Y(4)×Y(9)+K4×Y(1)+K4×Y(8)×Y(9)+   *    700    6
     +K3×Y(8)×Y(3)+A1×Y(29)+K5×Y(1)×Y(2)+K6×Y(2)+K2×Y(3)×Y(5);               *    800    7
Z(4)=D1×Y(3)+D2×Y(4)×Y(9)+D3×Y(4)×Y(9) +K7×Y(8)×Y(4)+K8×Y(3) ;               *    900    8
Z(5)=K3×Y(2)+E1×Y(2)×Y(9)+E2×Y(3)×Y(9)+E3×Y(4)×Y(9)+E4×Y(5)+E5×Y(5) ;        *   1000    9
Z(6)=F1×Y(2)×Y(9)+F2×Y(3)×Y(9)+F3×Y(4)×Y(9)+F4×Y(6) +F3×Y(6)+                *   1100   10
  F1×Y(12)×Y(8)+A1×Y(17)×Y(26)+K3×Y(12)×Y(19)+K1×Y(9)+K4×Y(13)×Y(29);         *   1200   11
Z(7)=G1×Y(6)+G2×Y(7)+G3×Y(7) ;                                               *   1300   12
Z(8)=H1×Y(1)  ;                                                              *   1400   13
Z(9)=I1×Y(10)+I2×Y(9)×Y(5)+I3×Y(9)×Y(7);                                     *   1500   14
Z(10)=J1×Y(10)+J2×Y(10)×Y(7)+J3×Y(5)×Y(10)+K2×Y(8)+6.578493748;              *   1600   15
Z(11)=K1×Y(10)+K3×Y(7)+K2×Y(9)×Y(7)+K3×Y(11) ;                               *   1700   16
Z(12)=C1×Y(3)×Y(9)+C2×Y(3)×Y(3)+C3×Y(3)+A1×  Y(9)+K4×Y(1)+K4×Y(8)×Y(9);     *   1800   17
Z(13)=C1×Y(3)×Y(9)+C2×Y(3)×Y(9)+C3×Y(3)+ K3 ×Y(9)+K4×Y(1)+K4×Y(8)×Y(9)      *   1900   18
     +K5×Y(1)×Y(2)+K6×Y(2)+K2×Y(3)×Y(5)+A2×Y(16)×Y(23)+B5×Y(8)×Y(14)         *   2000   19
 +C3×Y(20)×Y(15)+J1×Y(28)×Y(5)+A2×Y(22)+A4×Y(5)+K1×Y(8)×Y(2);                *   2100   20
Z(14)=D1×Y(3)+D2×Y(4)×Y(9)+D3×Y(4)×Y(9) +K7×Y(8)×Y(4)+K8×Y(3) ;             *   2200   21
Z(15)=K3×Y(2)+E1×Y(2)×Y(9)+E2×Y(3)×Y(9)+E3×Y(4)×Y(9)+E4×Y(5)+E5×Y(5) ;      *   2300   22
Z(16)=F1×Y(2)×Y(9)+F2×Y(3)×Y(9)+F3×Y(4)×Y(9)+F4×Y(6) ;                       *   2400   23
Z(17)=G1×Y(6)+A1×Y(7)×Y(8)+G2×Y(7)+G3×Y(7) ;                                 *   2500   24
Z(18)=H1×Y(1)  ;                                                             *   2600   25
Z(19)=J1×Y(10)+A3×Y(1)×Y(23)+J2×Y(10)×Y(7)+J3×Y(5)×Y(10)+ 4.08=02×Y(3)       *   2700   26
```

```
  + 5.344*Y(29)*Y(23)+ 3.141*Y(15)) ;                                    *   2800    27
Z(20)=K1*Y(10)*Y(7)+K2*Y(9)*Y(7)+K3*Y(11) ;                              *   2900    28
Z(21)=B3*Y(9)*Y(10)+K2*Y(27)+J1*Y(10)+I2*Y(9)*Y(5)+I3*Y(9)*Y(7)) ;       *   3000    29
Z(22)=C2*Y(3)*Y(9)+C2*Y(3)*Y(9)+C3*Y(3)*Y(4)+K4*Y(3)*Y(2)+K4*Y(8)*Y(9)) ;  *   3100    30
Z(23)=C2*Y(3)*Y(9)+C2*Y(3)*Y(9)+C3*Y(3)*Y(8)+K4*Y(2)*Y(25)+K4*Y(8)*Y(9)  *   3200    31
 +F1*Y(1)*Y(21)+K5*Y(2)*Y(2)+K6*Y(2)+K2*Y(3)*Y(5)+A1*Y(8)+F4*Y(9)*Y(14)  *   3300    32
     +C1*Y(8)*Y(9)+D2*Y(3)+D3*Y(2)+D6*Y(9)*Y(9)) ;                        *   3400    33
Z(24)=D2*Y(3)+D2*Y(8)*Y(9)+D3*Y(8)*Y(9) +K7*Y(8)*Y(8)+K8*Y(3) ;          *   3500    34
Z(25)=K3*Y(2)+F2*Y(2)*Y(9)+F2*Y(3)*Y(5)+F3*Y(8)*Y(9)+F4*Y(5)+F5*Y(5) ;   *   3600    35
Z(26)=F2*Y(2)*Y(9)+F2*Y(3)*Y(5)+F3*Y(6)*Y(9)+F4*Y(6) ;                   *   3700    36
Z(27)=G2*Y(6)+G2*Y(7)+G3*Y(7) ;                                          *   3800    37
Z(28)=H2*Y(2) ;                                                          *   3900    38
Z(29)=J2*Y(20)+J2*Y(20)*Y(7)+J3*Y(5)*Y(20)+A2*Y(8)+J1*Y(1)*Y(9)+ 4.0) ;  *   4000    39
Z(30)=I2*Y(20)+I2*Y(9)*Y(5)+I3*Y(9)*Y(7)) ;                              *   4100    40
Z0(1)= 118.0 ;                                                           *   4200    41
Z0(2)=  74.0 ;                                                           *   4300    42
Z0(3)=  37.0) ;                                                          *   4400    43
Z0(4)= 10900.0) ;                                                        *   4500    44
Z0(5)=  97.0  ;                                                          *   4600    45
Z0(6)=  29.0  ;                                                          *   4700    46
Z0(7)= 430.0  ;                                                          *   4800    47
Z0(8)=  0.0   ;                                                          *   4900    48
Z0(9)=  1.0   ;                                                          *   5000    49
Z0(10)= 4.0   ;                                                          *   5100    50
Z0(11) =  0.0 ;                                                          *   5200    51
Z0(12) = 3.5645 ;                                                        *   5300    52
Z0(13) = 34567.89 ;                                                      *   5400    53
Z0(14) = 0) ;                                                            *   5500    54
Z0(15) = 1) ;                                                            *   5600    55
Z0(16) = .0000005 ;                                                      *   5700    56
Z0(17) = 3.0098766555) ;                                                 *   5800    57
```

```
Z0(18) = .983 ;                                          *  5900   58
Z0(19) = 4.675;                                          *  6000   59
Z0(20) = 0;                                              *  6100   60
Z0(21) = 345.67452-1;                                    *  6200   61
Z0(22) = 3356345.0 ;                                     *  6300   62
Z0(23) = 3.0;                                            *  6400   63
Z0(24) = 0;                                              *  6500   64
Z0(25) = 0;                                              *  6600   65
Z0(26) = 0;                                              *  6700   66
Z0(27) = 0;                                              *  6800   67
Z0(28) = 0;                                              *  6900   68
Z0(29) = 0;                                              *  7000   69
Z0(30)= 0;                                               *  7100   70
A1= -1.0;                                                *  7200   71
A2= -2.0;                                                *  7300   72
A3=  .05;                                                *  7400   73
A4= 0.1 ;                                                *  7500   74
A5= 0.56 ;                                               *  7600   75
A6= 3.1415 ;                                             *  7700   76
B1 = 1.0;                                                *  7800   77
B2= -0.05;                                               *  7900   78
B3= -1.5;                                                *  8000   79
B4= -0.03;                                               *  8100   80
B5= -0.03;                                               *  8200   81
C1= -0.02;                                               *  8300   82
C2= -0.02 ;                                              *  8400   83
C3= -3.0  ;                                              *  8500   84
D1=  3.0  ;                                              *  8600   85
D2= -0.005 ;                                             *  8700   86
D3= -0.005 ;                                             *  8800   87
F1=  0.03  ;                                             *  8900   88
```

```
F2=  0.02  ;                                          *  9000   89
F3=  0.005 ;                                          *  9100   90
F4= -0.54  ;                                          *  9200   91
F5= -0.033 ;                                          *  9300   92
F1=  0.03  ;                                          *  9400   93
F2=  0.02  ;                                          *  9500   94
F3=  0.005 ;                                          *  9600   95
F4= -2.0   ;                                          *  9700   96
G1=  2.0   ;                                          *  9800   97
G2=-0.01       ;                                      *  9900   98
G3=-0.033      ;                                      *  10000  99
H1= 2.0        ;                                      *  10100  100
H2= 5.08497877 5;                                     *  10200  101
T2=-0.01000 ;                                         *  10300  102
T3=-0.00233 ;                                         *  10400  103
J1=-0.50       ;                                      *  10500  104
J2=-0.00058 ;                                         *  10600  105
J3=-0.00250 ;                                         *  10700  106
K1= 0.00058 ;                                         *  10800  107
K2= 0.00233 ;                                         *  10900  108
K3=-1.0 ;                                             *  11000  109
K4=-3.58497364;                                       *  11100  110
K5= -5456.485466;                                     *  11200  111
K6= -345.56770-04;                                    *  11300  112
K7= 45677.999 ;                                       *  11400  113
K8= 45.546859 ;                                       *  11500  114
K9= 4566.4902 ;                                       *  11600  115
TO = 0 ;                                              *  11700  116
TF =1;                                                *  11800  117
TP =0.0,0.25,0.5,0.75,1.0 ;                           *  11900  118
STEP =  0.05 ;                                        *  12000  119
```

```
END                                                                    *  12100      120
  0123454789£@?:>;+ABCDEFGHI.[<(++&JKLMNOPQR$*-);; /STUVWXY7,%
  1111111111111111111111111111111111111111111111111111111100000
  1111111111111111111111111111111111111111111111111111111110000
  1111111111111111111111111111111111111111111111111111111110000
PERCENT OF AVAILABLE STORAGE USED IS 94.35%


SPEED-UP FACTOR IS 55.7



CONGRATULATIONS,  NO SOURCE PROGRAM ERRORS WERE DETECTED IN PASS 1.
          THE TOTAL TIME WAS   1 MINUTES 19.6 SECONDS.

          TIME FOR INITIALIZATION:        0 MINUTES 13.2 SECONDS.
          TIME FOR SCANNING:              0 MINUTES 11.9 SECONDS.
          TIME FOR PARSING:               0 MINUTES 27.6 SECONDS.
          TIME FOR SEMANTIC ACTIONS:      0 MINUTES 26.9 SECONDS.
          TIME FOR ERROR RECOVERY:        0 MINUTES  0.0 SECONDS.
          TIME FOR DEBUG PRINTING:        0 MINUTES  0.0 SECONDS.

           120 CARDS READ AT   90 CARDS PER MINUTE.
```

APPENDIX B

ILLIAC IV ASSEMBLY LANGUAGE OUTPUT PRODUCED BY THE INTEGRATOR FOR THIS SAMPLE PROBLEM

```
                CODES  /RIGASK      LISTED BY SNAP    ON     APRIL 30, 1969  AT  20:39:44.8

      BEGIN                                                                    00 00100      1
% THE FOLLOWING PROGRAM WAS GENERATED BY THE GENERAL DIFFERENTIAL               00 00200      2
%                                                                              00 00300      3
% EQUATION SOLVER PROGRAM.  IT INTEGRATES THE INPUT EQUATIONS USING            00 00400      4
% A FOURTH ORDER RUNGE KUTTA METHOD.                                           00 00500      5
%                                                                              00 00600      6
      CMNS     1;%                                                             00 00700      7
%              ..........C( MEMORY DATA..,..........                           00 00800      8
ONE:  EQU      =400014000000000000000;8;% FLOATING POINT ONE                   00 00900      9
OSXTH: EQU     =400001252525252525252;8;% FLOATING ONE /SIXTH                  00 01000     10
HALF:  EQU     =400010400000000000000;8;% FLOATING ONE HALF                    00 01100     11
TWO:   EQU     =400024000000000000000;8;% FLOATING TWO                         00 01200     12
TAU:   EQU     =400016000000000000000;8;% TIME DELAY(1.5)                      00 01300     13
,BOUND:EQU     $145;%            LEFT RIGHT ROUTING BOUND                     00 01400     14
,COUNT:EQU     $146;%            NUMBER OF ITERATIONS                         00 01500     15
,INC:  EQU     $147;%            NUMBER OF VALUES TO BE STORED                00 01600     16
,TEMP2:EQU     $148;%                                                         00 01700     17
,TEMP3:EQU     $149;%                                                         00 01800     18
,LEFT: EQU     $150;%            LEFT ROUTE LIMIT                             00 01900     19
,MASK: EQU     $152;%            MASK                                         00 02000     20
%              ..........BE MEMORY DATA.............                          00 02100     21
ARITH: BLL     128;                                                           00 02200     22
       BLK     20;                                                            00 02300     23
EXPSV: BLL     128;                                                           00 02400     24
       BLK     11;                                                            00 02500     25
XS:    BLL     128;%                                                          00 02600     26
       BLK     55;%                                                           00 02700     27
       BLL     128;%                                                          00 02800     28
       BLK     200;%                                                          00 02900     29
SUM:   BLK     1;%               RUNGE KUTTA CONSTANTS                        00 03000     30
SUM1:  BLK     1;%                                                            00 03100     31
SUM2:  BLK     1;%                                                            00 03200     32
SUM3:  BLK     1;%                                                            00 03300     33
       BLL     128;%                                                          00 03400     34
INDEX: DATA    0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,         00 03500     35
       DATA    21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37;            00 03600     36
       DATA     38,39,40,41,42,43,44,45,46,47,48,49,50,51,52;                 00 03700     37
       DATA    53,54,55,56,57,58,59,60,61,62,63;                              00 03800     38
XK:    BLK     1;%               VALUE FROM LAST STEP                         00 03900     39
XKTEM: BLK     1;%               INTERMEDIATE VALUES OF X                     00 04000     40
FRAC:  BLK     1;%               PARAMETERS TO ACCESS XT(T-TAU)               00 04100     41
INTE:  BLK     1;%                                                            00 04200     42
       BLL     128;%                                                          00 04300     43
       BLK     10;%                                                           00 04400     44
TSUM:  BLK     10;%     SCRATCH STORAGE FOR FORMING SUMS                      00 04500     45
INDX1: EQU     =1%63=0;%                                                      00 04600     46
ENB:   EQU     =177777777777777777777;8;%  ENABLE BITS                       00 04700     47
CONST:                                                                        00 04800     48
       DATA                                                                   00 04900     49
              3.1415,0.1,-1.5,-0.03,-0.02,-1.0,-1.0,-345.5677=-04,            00 05000     50
              -1.005,45.566659,0.02,-0.033,0.005,0.03,0.00058,                00 05100     51
              -0.01,,-0.50,0.00233,-1.0,-0.02,-1.0,-0.02,-1.5,                00 05200     52
              -5456.48584,-2.0,3.0,0.00058,-0.005,-1.0,0.005,0.03,            00 05300     53
              -2.0,-0.01,-0.50,-0.00250,3.1416,-1.0,,-0.02,                   00 05400     54
              -3.58697364,-0.02,-3.58697364,-345.5677=-04,-0.56,              00 05500     55
              -0.005,-0.005,45.566659,0.02,-0.033,0.005,-0.01,                00 05600     56
              -0.00058,-2.0,-0.01000,0,0,0,0.0;                               00 05700     57
```

```
 FILL     128;                                                      00 05800
 DATA                                                               00 05900
          -1.0,.05,600,.-1.5,0.00233,-3.0,-3.58697364,             00 06000
          -5456.48586,3.0,45677.999,0.03,-0.56,0.02,0.005,-1.0,    00 06100
          2.0,2.0,-0.00233,-0.0250,0.00058,-1.0,-3.0,              00 06200
          -3.58697364,-3.0,-3.58697364,0.00233,-3.0,0.1,-0.005,    00 06300
          45.566859,0.02,-0.03,0.005,-1.0,2.0,-0.00058,5.346,      00 06400
          0.00233,0.00233,-0.00233,-3.0,-0.02,-3.58697364,         00 06500
          -5456.48586,-1.0,-0.005,-0.005,45677.999,0.02,-0.56,     00 06600
          0.02,-0.01,5.0469/3775,-0.00250,4.0,-0.00233,0,0,0,0,    00 06700
          0:                                                        00 06800
 FILL     128:                                                      00 06900
 DATA                                                               00 07000
          0.00058,-2.0,0.56,-0.05,-0.03,-0.02,-3.58697364,-1.0,    00 07100
          0.00233,-0.005,-1.0,0.005,0.03,-2.0,-1.0,-3.58697364,    00 07200
          -0.033,-0.01000,-0.00058,6.5769374,0.00233,-0.02,         00 07300
          -3.58697364,-0.02,-3.58697364,-345.56778-04,-0.03,        00 07400
          -2.0,3.0,45677.999,0.03,-0.56,0.02,2.0,-0.033,.05,        00 07500
          4.06-02,0.00058,-1.5,-0.01000,-0.02,-3.58697364,-3.0,     00 07600
          0.03,0.00233,-0.02,-0.005,-1.0,0.005,0.02,-2.0,          00 07700
          -0.033,-0.00058,-0.56,-0.01000,0,0,0,0,0;                00 07800
 FILL     128;                                                      00 07900
 DATA                                                               00 08000
          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,        00 08100
          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,        00 08200
          0,0,0,0,0,0,0,0,0:                                        00 08300
 FILL     128;                                                      00 08400
TAGS:                                                               00 08500
 DATA                                                               00 08600
           3,  0,  1,  2,  3,  1,  8,  0,  4,  1,  3,  0,          00 08700
           4, 12,  0,  1,  0,  0,  1,  0,  3,  1,  3,  0,          00 08800
           1, 16, 28,  8,  0,  1,  0,  2,  0,  0,  0,  5,          00 08900
           0,  0,  0,  3,  3,  3,  8,  1,  9,  0,  4,  0,          00 09000
           3,  1,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 09100
           1,  0:                                                   00 09200
 FILL     128:                                                      00 09300
 DATA                                                               00 09400
          30,  7,  9,  2,  9,  2,  3,  2,  9,  3,  9,  5,          00 09500
           1,  8,  9,  7, 10, 10,  8,  7,  9,  9,  9,  9,          00 09600
           2, 23,  5,  2,  0,  2,  9,  0,  6,  7, 10, 10,          00 09700
          31, 11, 10,  9,  2,  9,  9,  2, 14,  2,  9,  3,          00 09800
           9,  5,  9,  7, 20,  4, 20,  0,  0,  0,  0,  0,          00 09900
           1,  1:                                                   00 10000
 FILL     128:                                                      00 10100
 DATA                                                               00 10200
           1, -1,  0, -1,  0, -1, -2, -3, -2, -3, -2, -3,          00 10300
          -2, -3, -4, -3,  0,  1,  0,  1,  2,  1,  2,  1,          00 10400
           0, -1, -2, -3, -2, -1, -2, -1, -2, -1,  2,  1,          00 10500
           1,  1,  2,  3,  2,  3,  2,  1,  0, -1,  0, -1,          00 10600
           0, -1,  1,  1,  4,  3,  4,  3,  0,  0,  0,  0,          00 10700
           1,  0:                                                   00 10800
 FILL     128:                                                      00 10900
 DATA                                                               00 11000
           0,  1,  0,  0,  0,  0,  8,  1,  0,  8,  2,  0,          00 11100
           3,  0, 12,  0,  0,  9,  5,  0,  0,  0,  8,  0,          00 11200
           8,  3, 20,  0,  4,  0,  3,  0,  4,  7,  0, 10,          00 11300
          27,  9,  0,  9,  3,  3,  2,  2,  0,  0,  0,  8,          00 11400
           2,  0,  3,  0,  0,  5,  0,  9,  0,  0,  0,  0,          00 11500
           0,  0:                                                   00 11600
 FILL     128;                                                      00 11700
 DATA                                                               00 11800
```

```
                    1,  2,  0,  2,  5,  3,  9,  2,  3,  4,  9,  5,          00 11900   119
                    9,  6, 19,  6,  1,  7, 10, 10, 11,  3,  9,  3,          00 12000   120
                    9,  5, 15,  8,  9,  3,  9,  5,  9,  8,  1,  7,          00 12100   121
                   23,  7, 27,  7,  4,  9, 25,  2,  8,  3,  3,  4,          00 12200   122
                    3,  5,  9,  6,  2, 20,  0,  7,  0,  0,  0,  0,          00 12300   123
                    0,  0;                                                  00 12400   124
          FILL    128;                                                      00 12500   125
          DATA                                                              00 12600   126
                    0, -1, -2, -1, -2, -1, -2, -3, -2, -3, -2, -3,          00 12700   127
                   -3, -3, -4, -3, -2, -1,  0,  1,  0,  1,  0,  1,          00 12800   128
                    1, -1, -2, -3, -2, -3, -2, -3, -2, -1,  0,  1,          00 12900   129
                    0,  1,  2,  1,  2,  3,  2,  1,  0, -1,  0, -1,          00 13000   130
                    0, -1,  0,  1,  2,  3,  2,  3,  0,  0,  0,  0,          00 13100   131
                    0,  0;                                                  00 13200   132
          FILL    128;                                                      00 13300   133
          DATA                                                              00 13400   134
                    0,  0,  0,  0,  2,  3,  0,  0,  3,  4,  0,  4,          00 13500   135
                    2,  0, 17, 13,  0,  9, 10,  0,  9,  3,  0,  3,          00 13600   136
                    0,  0,  8,  0,  0,  8,  2,  0,  3,  0,  0,  1,          00 13700   137
                    0, 10,  9,  9,  3,  8,  3,  1,  3,  8,  9,  4,          00 13800   138
                    1,  4,  2,  0,  0, 20,  1,  9,  0,  0,  0,  0,          00 13900   139
                    0,  0;                                                  00 14000   140
          FILL    128;                                                      00 14100   141
          DATA                                                              00 14200   142
                    8,  1,  5,  2,  9,  9,  1, 29,  5,  9,  2,  9,          00 14300   143
                    9,  6, 24, 29,  7,  5,  7,  0,  7,  9,  1,  9,          00 14400   144
                    1,  2, 14, 22,  3,  4,  9,  5,  9,  6,  7, 23,          00 14500   145
                    3,  7, 10,  5,  9,  9,  8, 21,  5,  9,  0,  9,          00 14600   146
                    2,  9,  9,  6,  7,  7,  9,  5,  0,  0,  0,  0,          00 14700   147
                    0,  0;                                                  00 14800   148
          FILL    128;                                                      00 14900   149
          DATA                                                              00 15000   150
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 15100   151
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 15200   152
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 15300   153
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 15400   154
                    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,          00 15500   155
                    0,  0;                                                  00 15600   156
          FILL    128;                                                      00 15700   157
INCS:      EQU    =1,       0,0; %                                         00 15800   158
COUNS:     EQU   =1.       20,0;                                          00 15900   159
DELT:     DATA   (0,05                          )64;                      00 16000   160
TO:       DATA   (0.                            )64;                      00 16100   161
LEFTS:     EQU    = 4; %                                                  00 16200   162
BOUNS:     EQU   =1. 9.0; /                                               00 16300   163
          BLK     2; %                                                    00 16400   164
XT:       DATA                                                            00 16500   165
          118,0.0,74.0,0,37.0,0,10900.0,0.0,97.0,0,29,0,0,                00 16600   166
          430.0,0,0.0.0,0;1.0,0,4.0,0,0,0,0,3.5645,0,34567.89,0,          00 16700   167
          0.,0,1.,0,.0000005,0,3.0098/6655,0,,9@3,0,4.675,0,              00 16800   168
          0.,0,345.6745@-1.0,3356345.0,0.3.0,0,0.,0,0.,0,0.,0,            00 16900   169
          0.,0,0.,0,0.,0.0.,0,0;                                          00 17000   170
          BLK     20; %                                                   00 17100   171
%                                                                         00 17200   172
START:  SUIT(1)   =63;%                                                   00 17300   173
        STL(1)    ,N63;%              USED AS MASK                        00 17400   174
        LIT(0)    =ENB;%                                                  00 17500   175
        LDEF1(0)              ;%                                          00 17600   176
        LIT(0)    =1#3#0;%          =1#LEFT*WIGHT#0                     00 17700   177
        STL(0)    ,BOUND;%                                                00 17800   178
        LIT(0)    =ONE;%                                                  00 17900   179
```

```
        LDA     $C0;%                                                   00 18000        180
        STA     XS;%                                                    00 18100        181
        LDA     XT;%                                                    00 18200        182
        STA     XT=1;%                                                  00 18300        183
        STA     XK;%                                                    00 18400        184
        STA     XKTEM;%                                                 00 18500        185
%.....................END OF DATA LOADING.....................          00 18600        186
        LIT(3)  =COUNS;                                                 00 18700        187
        STL(3)  .COUNT;                                                 00 18800        188
% THIS NEXT SECTION IS USED IF A DELAY IS USED. (X(T-TAU) )             00 18900        189
%  THE FOLLOWING PART OF THE PROGRAM IS THE RUNGEKUTTA CALCULATIONS     00 19000        190
NEWT:   STL(2)  .TEMP2;%        ******************************          00 19100        191
        LIT(1)  =HALF;%                                                 00 19200        192
        STL(3)  .TEMP3;%                                                00 19300        193
        SLIT(2) =EVAL;%         CALL SUBROUTINE TO EVALUATE F(X,T).     00 19400        194
        EXCHL(2) $ICR;%          SUBROUTINE BRANCH                      00 19500        195
        SKIP    ,0;%                                                    00 19600        196
        LDA     DELT;%          T:=T+DELT#2(T+ DELT) =                  00 19700        197
        MPRN    $C1;%                                                   00 19800        198
        ADRN    T0;%                                                    00 19900        199
        STA     T0;%                                                    00 20000        200
        LDA     SUM;%                                                   00 20100        201
        STA     SUM1;%                                                  00 20200        202
        MPRN    $C1;%           XKTEM:=XK+DELT*K1#2                     00 20300        203
        MPRN    DELT;%                                                  00 20400        204
        ADRN    XK;%                                                    00 20500        205
        STA     XKTEM;%                                                 00 20600        206
        SLIT(2) =EVAL;%         SUBROUTINE CALL TO FIND K2              00 20700        207
        EXCHL(2) $ICR;%                                                 00 20800        208
        SKIP    ,1;%                                                    00 20900        209
        LDA     SUM;%           XKTEM:= XK + DELT*K2#2                  00 21000        210
        STA     SUM2;%                                                  00 21100        211
        MPRN    DELT;%                                                  00 21200        212
        MPRN    $C1;%                                                   00 21300        213
        ADRN    XK;%                                                    00 21400        214
        STA     XKTEM;%                                                 00 21500        215
        SLIT(2) =EVAL;%         SUBROUTINE CALL TO GET K3               00 21600        216
        EXCHL(2) $ICR;%                                                 00 21700        217
        SKIP    ,1;%                                                    00 21800        218
        LDA     SUM;%           XKTEM:= XK + DELT*K3                    00 21900        219
        STA     SUM3;%                                                  00 22000        220
        MPRN    DELT;%                                                  00 22100        221
        ADRN    XK;%                                                    00 22200        222
        STA     XKTEM;%                                                 00 22300        223
        LDA     DELT;%                                                  00 22400        224
        MPRN    $C1;%                                                   00 22500        225
        ADRN    T0;%                                                    00 22600        226
        STA     T0;%                                                    00 22700        227
        SLIT(2) =EVAL;%         SUBROUTINE CALL TO GET K4               00 22800        228
        EXCHL(2) $ICR;%                                                 00 22900        229
        SKIP    ,0;%                                                    00 23000        230
        LDA     SUM2;%          XK:=XK+(1#6)*DELT(K1+2*K2+2*K3+K4)      00 23100        231
        ADRN    SUM3;%                                                  00 23200        232
        LIT(2)  =TWO;%                                                  00 23300        233
        MPRN    $C2;%                                                   00 23400        234
        ADRN    SUM1;%                                                  00 23500        235
        ADRN    SUM;%                                                   00 23600        236
        MPRN    DELT;%                                                  00 23700        237
        LIT(1)  =SIXTH;%                                                00 23800        238
        MPRN    $C1;%                                                   00 23900        239
        ADRN    XK;%                                                    00 24000        240
```

```
        STA         XK];%                                                   00 24100    241
        STA         >KTEM];%                                                00 24200    242
        LDL(2)      ,TEMP2];%                                               00 24300    243
        TYLTM(2)    ,NEWT];%      IF NOT TO BE SAVED STEP                   00 24400    244
        STA         XT(3)];%                                                00 24500    245
        LDL(2)      ,INC];%                                                 00 24600    246
        LDL(3)      ,TEMP3];%                                               00 24700    247
        TYLTM(3)    ,NEWT];%      LOOP UNTIL END                            00 24800    248
        JUMP        60];%          ****************************************** 00 24900    249
        SKIP        ,0];%                                                   00 25000    250
% END OF RUNGE KUTTA CALCULATIONS ROUTINE                                   00 25100    251
%                                                                           00 25200    252
% BEGINNING OF FUNCTION EVALUATION SUBROUTINE.                              00 25300    253
FVAL:   EXCHL(2)    $02];%              SAVE CARS                           00 25400    254
        EXCHL(3)    $03];%                                                  00 25500    255
        STL(0)      $011];%                                                 00 25600    256
        STL(1)      $012];%                                                 00 25700    257
        SUBT(2)     =ROTE];%            CALL ROTE                           00 25800    258
        EXCHL(2)    $ICR];%                                                 00 25900    259
        SKIP        ,0];%                                                   00 26000    260
        LDA         =0];%                                                   00 26100    261
        STA         SUM];%              ZERO SUM                            00 26200    262
FFIJ:   STL(2)      $07];%           ********************************       00 26300    263
        LDL(2)      ,ROUND];%         RANGE OF ROUTES LOADED                00 26400    264
        LDA         =0];%             ZERO OUT TSUM LOCATIONS               00 26500    265
INIT:   CSUB(2)     ,LEFT];%                                                00 26600    266
        STA         TSUM(2)];%                                              00 26700    267
        CADD(2)     ,LEFT];%                                                00 26800    268
        TYLTM(2)    ,INIT];%                                                00 26900    269
          LDA        CONST+  0;                                             00 27000    270
          LDX        TAGS+   0;                                             00 27100    271
          MLRN       *XS   ;                                                00 27200    272
          LDX        TAGS+   1;                                             00 27300    273
          MLRN       *XS   ;                                                00 27400    274
          LDX        TAGS+   2;                                             00 27500    275
          ADRN       *TSUM;                                                 00 27600    276
          STA        *TSUM;                                                 00 27700    277
          LDA        CONST+  1;                                             00 27800    278
          LDX        TAGS+   3;                                             00 27900    279
          MLRN       *XS   ;                                                00 28000    280
          LDX        TAGS+   4;                                             00 28100    281
          MLRN       *XS   ;                                                00 28200    282
          LDX        TAGS+   5;                                             00 28300    283
          ADRN       *TSUM;                                                 00 28400    284
          STA        *TSUM;                                                 00 28500    285
          LDA        CONST+  2;                                             00 28600    286
          LDX        TAGS+   6;                                             00 28700    287
          MLRN       *XS   :                                                00 28800    288
          LDX        TAGS+   7;                                             00 28900    289
          MLRN       *XS   ;                                                00 29000    290
          LDX        TAGS+   8;                                             00 29100    291
          ADRN       *TSUM;                                                 00 29200    292
          STA        *TSUM;                                                 00 29300    293
        LDA         =0];%             ZERO A REGISTERS                      00 29400    294
        LDL(3)      ,ROUND];%         ROUTING ROUND                         00 29500    295
FINS:   LDL(2)      $03];%                                                  00 29600    296
        CSUB(2)     ,LEFT];%          SUB LEFT MAX TO GET NEGATIVE ROUTES   00 29700    297
        LDR         TSUM(2)];%        RGR= PARTIAL SUMS                     00 29800    298
        CAND(2)     ,N63];%           MUST DO ROUTE OF 63 INSTEAD OF -1     00 29900    299
        RTL         0(2)];%           ROUTE PARTIAL SUMS TO CORRECT PF      00 30000    300
        ADRN        $R];%             ADD TO A REGISTERS                    00 30100    301
```

```
        TXLTM(3)  ,FINS!%       LOOP,ROUTE RIGHT 1 PLUS LAST                00 30200   302
        ADRN      SUM!%                                                     00 30300   303
        STA       SUM!%         RESULT IN SUM                               00 30400   304
        LDL(2)    $07!%                                                     00 30500   305
        EXCHL(3)  $03!%                                                     00 30600   306
        EXCHL(2)  $02!%                                                     00 30700   307
        LDL(0)    $011!%                                                    00 30800   308
        LDL(1)    $012!%                                                    00 30900   309
        EXCHL(2)  $ICR!%        ******************************              00 31000   310
        SKIP      ,0;%                                                      00 31100   311
% END OF FUNCTION EVALUATION SUBROUTINE.                                    00 31200   312
%                                                                           00 31300   313
ROTE!   LDX       XKTEM!%                                                   00 31400   314
        SETF      E.OR.-F!%       THIS TAKES THE VALUES STORED              00 31500   315
        LDA       INDEX!%         ACROSS THE PF,S IN XKTEM                  00 31600   316
        LTT(1)    =INDX1!%        AND DISTRIBUTES THEM INTO THE             00 31700   317
ROTX!   LDX       $A1%            64 LOCATIONS FOLLOWING XK.                00 31800   318
        STR       *XS+1!%                                                   00 31900   319
        RTL       5!%                                                       00 32000   320
        LDA       $A1%                                                      00 32100   321
        ADM       =1!%                                                      00 32200   322
        AND       =63!%                                                     00 32300   323
        TXLTM(1)  ,ROTX!%                                                   00 32400   324
        LTT(0)    =ENB!%                                                    00 32500   325
        LDEF1(0)            !%                                            00 32600   326
        EXCHL(2)  $ICR!%                                                    00 32700   327
        SKIP      ,0!%          ******************************            00 32800   328
%                                                                           00 32900   329
% BEGINNING OF ROUTINE TO RECOVER VALUES IS DELAY IS USED.                  00 33000   330
TAUR!   LTT(3)    =77777771B1!%                                            00 33100   331
        LDL(0)    .TEMP2!%                                                  00 33200   332
        CAND(0)   $C3!%           AND OUT LAST 24 BITS OF COUNTER           00 33300   333
        LTT(3)    =0400600000000000000000181%                               00 33400   334
        CDR(0)    $C3!%           CHANGE TO FLOATING POINT                  00 33500   335
        LDL(3)    .TEMP3!%                                                  00 33600   336
        LDX       $C3!%            GET COUNTER INDICATING STORED VALUES     00 33700   337
        XO        INTF!%                                                    00 33800   338
        TYG       =10000018!%      TEST TO SEE IF NEGATIVE                  00 33900   339
        SETF      1.AND.F!%        IF SO USE INITIAL VALUE                  00 34000   340
        LDA       XT;%                                                      00 34100   341
        STA       XKTEM!%                                                   00 34200   342
        SETF      -E.AND.F;%                                                00 34300   343
        SETF      -T.OR.F!%        IF NOT FETCH                             00 34400   344
        LDA       FRAC;%           MULT PRECEDING VALUE BY FRAC             00 34500   345
        ADRN      $C0!%              PLUS COUNTER FRACTION IN CARO          00 34600   346
        MURN      *XT-1;%                                                   00 34700   347
        LDS       $A1%             MULT INTF VALUE BY 1-(FRAC               00 34800   348
        LTT(3)     =ONE!%                          +COUNTER FRACTION)       00 34900   349
        LDA       $C3;%                                                     00 35000   350
        SBRN      FRAC!%                                                    00 35100   351
        SBRN      $C0!%                                                     00 35200   352
        MURN      *XT!%                                                     00 35300   353
        ADRN      $S1%             ADD AND STORE IN XKTEM                   00 35400   354
        STA       XKTEM!%                                                   00 35500   355
        LTT(0)    =ENB!%                                                    00 35600   356
        LDEF1(0)            !%    RE-ENABLE AND RETURN                      00 35700   357
        EXCHL(2)  $ICR!%                                                    00 35800   358
        SKIP      ,0!%            ******************************            00 35900   359
GO!      SKIP     ,0;%                                                      00 36000   360
        END START .                                                         00 36100   361
```

APPENDIX C

BNF INPUT SPECIFICATIONS FOR THE INTEGRATOR

(INPUT TO THE TWS SYSTEM ALONG WITH SEMANTICS
TO PRODUCE THE INTEGRATOR-TRANSLATOR. THE
SEMANTICS ARE TOO LONG TO BE LISTED HERE)

```
                GOOFS  /SYNTAX      LISTED BY SNAP    ON     APRIL 30, 1969  AT  20:39:43.3

/GOOFS  ;                                                                      00 00100              1
PRINT ALL TABS;                                                                00 00200              2
FPARSER;                                                                       00 00300              3
    #Y,T,Z,LIST,TF,TO,END,LOG,TO,TP,COS,EXP,VARSTEP,RUNGE,SIN,EXTRAP;          00 00400              4
<PROG>::= <STATE>  #5 / <PROG> > <SEP> <STATE> #5;                             00 00500              5
<SEP> ::=  #1 / <>;                                                            00 00600              6
<STATE> ::= <DIFFQN> #10/ <INIT> #15 / <TINIT> #20 / <TFINAL> #25/             00 00700              7
            <TPRINT> #30/ <CONSTDEC> #35 /<CONTROL> #40/                       00 00800              8
   #STEP =<REALNUM> #41/ #END #1/ #ERROR = <REALNUM> #61 ;                     00 00900              9
<CONTROL> ::= ## <CONWRD> #44/<CONTROL> <CONWRD> #43 ;                         00 01000             10
<CONWRD> ::=  #DEBUG #42/ #TEST #39/   #CORPRE   #38/  #RUNGE #37 /            00 01100             11
          #EXINSERT = <*N> #4/ / #CODEINSERT =  <*N> #45/ #FUNMODE #57/        00 01200             12
           / #ERRORCHECK  #58 / #CONSTDEF #59/ #PATTERN #60;                   00 01300             13
<DIFFQN>::= #Z( <INDEX> ) #3  = <EXP>;                                         00 01400             14
<INDEX>::= <*N> #36 ;                                                          00 01500             15
<EXP> ::=  <TERM> #34 /<ADP> <TERM> #33  / <EXP> <ADP> <TERM> #32 ;            00 01600             16
<TERM>::= <FACTOR> #31/  <TERM> <MOP> <FACTOR> #20;                            00 01700             17
<ADP> ::=  + #28 /- #27  ;                                                     00 01800             18
<FACTOR> ::= <PRIMARY> #26  /<FACTOR> * #48  <PRIMARY> #24  ;                  00 01900             19
<MOP> ::=  x #23 /#/#22 ;                                                      00 02000             20
<PRIMARY>::=<REALNUM> #21/<*1> #19/#Y(<INDEX>) #18  /#T #17 /<FUNCT>#16        00 02100             21
             / ( #2 <EXP>)  #14  ;                                             00 02200             22
<REALNUM>::=  <*N> #49 <FRACTION> <EXPONENT> #54/ . <*N> #56                   00 02300             23
                   <EXPONENT> #54;                                             00 02400             24
<FRACTION>::= .<*N> #50/ .  / <> ;                                             00 02500             25
<EXPONENT>::= #@ <EXPSIGN> <*N> #51/<>;                                        00 02600             26
<SIGN>::= + / - #55/ <>;                                                       00 02700             27
<EXPSIGN>::= +/- #52/<>;                                                       00 02800             28
<FUNCT> ::=  <FUNID> (<EXP>) #13 ;                                             00 02900             29
<FUNID> ::= #SIN #12 / #COS #11 / #LOG #9 / #EXP #8 ;                          00 03000             30
<INIT>::=#Z0(<INDEX>)= <SIGN><REALNUM>;                                        00 03100             31
<TINIT>::=#T0= <SIGN><REALNUM>;                                                00 03200             32
<TPRINT>::= #TP=<SIGN> <REALNUM> #46/ <TPRINT>, <SIGN> <REALNUM> #7/           00 03300             33
      #TP = (<REALNUM> )  #6 ;                                                 00 03400             34
<TFINAL>::=#TF = <SIGN> <REALNUM>;                                             00 03500             35
<CONSTDEC>::= <*T> #4  = <SIGN> <REALNUM> ;                                    00 03600             36
                                                                               00 03700             37
```

LIST OF REFERENCES

[1] Barnes, G. H., et al., The ILLIAC IV Computer, IEEE Transactions on Computers, C-17, 8 (August, 1968) pp. 746-757.

[2] Gear, C. W., The Numerical Integration of Ordinary Differential Equations of Various Orders, Argonne National Laboratory, ANL - 7126, January 1966.

[3] McCarthy, T. E., Solution of Ordinary Differential Equations Related to Metabolic Systems, Department of Computer Science, University of Illinois, July 1968, ILLIAC IV Document No. 197.

[4] McCracken, Daniel and Dorn, W., Numerical Methods and Fortron Programming, John Wiley and Sons, pp. 317-329, New York 1964.

[5] Nordsieck, Arnold, On Numerical Integration of Ordinary Differential Equations, Mathematics of Computation, 16 (77-80), pp. 22-49 (1962).

[6] Rosen, Saul, Programming Systems and Languages, McGraw-Hill Book Company, New York 1967.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

AN AUTOMATIC INTEGRATOR FOR ORDINARY DIFFERENTIAL EQUATIONS FOR ILLIAC IV

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Thomas Edmund McCarthy

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1, 1969	57	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
46-26-15-305	
b. PROJECT NO. US AF 30(602)4144	DCS Report No. 336
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DCS.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
NONE	Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

13. ABSTRACT

This report describes the design and implementation of an automatic integrator for systems of ordinary differential equations for use on the ILLIAC IV computer system. This integrator accepts a simple statement of the differential equations and their initial values and produces an ILLIAC IV Assembly Language code which can be used to solve them. The intent is to produce a code which will be efficient on a large parallel computer.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ordinary differential equations						
automatic integration						
parallel computation						
ILLIAC IV						